TOWARD OPEN NETWORK DATA-EXCHANGE PROTOCOLS FOR CONSTRUCTION METROLOGY AND AUTOMATION: LIVEVIEW

Lawrence E. Pfeffer, Ph.D. and DeWitt T. Latimer IV

Construction Automation and Metrology Group, NIST
100 Bureau Drive, Stop 8611
Gaithersburg, MD 20899-8611

pfeffer@nist.gov dlatimer@nist.gov

Abstract: A significant problem in construction automation is the difficulty of interfacing and integrating subsystems (e.g. sensors, databases, visualization systems) into a useful whole. In this paper, we describe our past and present efforts toward the design and implementation of network protocols to address this problem. Our current design effort concentrates on communicating, "who, what, where, and when" data over internet-protocol (IP) networks, and is based on the IEEE 1278 (distributed interactive simulation) protocol family, with some extensions to meet the anticipated needs of construction automation systems.

Keywords: Automation, Construction, Data-Exchange, Distributed, Metrology, Protocol.

1. INTRODUCTION

One of the significant problems facing developers of construction metrology and automation systems is the difficulty of interfacing to a wide variety of subsystems and integrating them into a useful whole. This is especially true, since the subsystems are often numerous, and the "best" for any task can change rapidly. This problem has both hardware and software aspects. Some of the hardware aspects were addressed in [6]; this paper concentrates on the software aspects.

At present, there is no generally accepted standard for software subsystems in a construction-automation system to use for exchanging data with one another. As a result, significant effort in implementing such systems goes into creating different software interfaces for each sensor, actuator, GUI, database, etc. Since there is often little prospect of interface-software reuse (because there is so little standardization), system developers are often hesitant to change, upgrade or experiment with new subsystems. They have been caught in an interface straightjacket.

We assert that, if we are to avoid this productivity sink in the future, we need some standard, broadly accepted means to communicate information between construction-automation software subsystems. Ideally, this interface should work whether the subsystems reside on the same computer, or on different ones.

A completely general distributed data-exchange system is probably too ambitious a goal. We believe that an attractive approach is to base a data-exchange system for construction-automation on an existing protocol (namely IEEE 1278 [3]), with some modifications and extensions relevant to construction automation. Our immediate goal is to design and implement a draft protocol (called LiveView) that concentrates on communicating, "who, what, where, and when" data in an efficient manner among construction-automation subsystems.

This paper starts with an overview of distributed communications software, and then outlines our earlier effort and some of the lessons learned. We then give an overview of the IEEE-1278, the distributed interactive simulation protocol -- the basis for LiveView. Finally, we outline some of the key features and design decisions of LiveView, and illustrate with examples from related work at NIST's National construction-automation test-bed (NCAT).

2. RELATED WORK

Since the field of communications in distributed (computer) systems is a vast one, we

will make do with an overview. There are both general-purpose approaches to distributed communications, e.g., DCOM (Microsoft) and CORBA [2], and there are also tools more-or-less specialized for sensors [4], robotics [5] and distributed simulations [3]. Among general-purpose distributed communications, there are three common models of information exchange: shared memory, request/response, and publish/subscribe (see [5] for a literature review and discussion.) There are also several different approaches to the types and flexibility of data communicated, ranging from pre-defined messages to self-describing data. Finally, there is a range of management approaches, ranging from centrally managed to completely distributed.

In the shared-memory model, each subsystem "sees" a global, shared memory that can be read or written to by any of the sub-systems that participate. This model is convenient, because it permits individual subsystems to treat the problem largely in a non-distributed fashion, and is basically symmetric. (The symmetry encompasses one-to-one, one-to-many, or many-to-many communications.) However, specialized hardware or software techniques (e.g. MUTEXes) are needed to ensure consistent replication of the shared memory among all users, and may not be sufficiently efficient or predictable for use in real-time systems. Another undesirable characteristic is that this model does not provide an easy, efficient means of being notified when some data in the shared memory is changed.

In request/response systems, information is exchanged by issuing individual requests and awaiting the responses. This model of information exchange (also called client/server) is asymmetric, and takes place between specific, individual sub-systems. This model is probably the most widely implemented, and has been used as the basis for file systems, windowing systems, and distributed databases, [5]. There are several limitations of this structure. First, it requires two transactions per update (the request and the response), and is not well suited for disseminating data in a one-to-many (or many-to-many) fashion. Secondly, since it is basically a pull-oriented system, the information flow is analogous to polling, with the requestor controlling the timing, and the respondent unable to send "alerts" that something has changed until/unless the requestor makes the analogous request.

In publish/subscribe systems, data providers publish their willingness to provide specific data, and subscribers register with publishers for the data they need. Once a subscription is arranged, the subscriber receives data updates from a publisher automatically. (This is analogous to web-based "push" systems, where the publisher initiates the update events.) This model is attractive for information that needs to be distributed to many receivers, and has been extended to many-to-many systems, see [5]. It is also attractive for event-driven systems, since the data-updates are events that can be acted upon without the overhead of polling. Closely related are broadcast/multicast systems, in which publishers send out information to every computer involved and do not keep track of specific subscribers. This approach is low overhead for broadcasters, but gives the receivers the burden of sifting through the broadcast messages for those of interest. The large-scale feasibility of this variant depends on a low-overhead multicast mechanism, such as UDP multicast.

At the risk of drastic over simplification, there are three basic strategies for message-based data representation: Fixed message types, pre-described data, and self-describing data. These strategies trade off efficiency against flexibility.

Fixed message systems have a finite vocabulary of message types, and are usually highly efficient, but are fundamentally limited by their rigidity. However, the rigid format facilitates simple, efficient consistency checking. (Note, however that if one of the "fixed" types is other/user-defined, both the fixed nature and the efficiency are be compromised.)

The pre-described data strategy is more flexible than the fixed-message approach, as it lets users pre-define their own data types at compilation time, using a data-description language such as IDL [2]. However, with the flexibility of user-pre-defined data comes both a loss of efficiency and a potential for inconsistency. Since the data-descriptions must be pre-defined (usually at compile time), there is a potential for inconsistency between sub-systems, e.g., if implemented by different groups. Run-time consistency checking is possible, but is not (generally speaking) automatic.

Finally, there is the self-describing data approach, in which every piece of data carries with it a self-description. This approach is extremely flexible, but generally has the most overhead associated with it.

3. EARLIER NIST WORK

Our initial foray into the distributed data-exchange (called tetSock) was driven by an application. We had a pre-existing robotic system (a 6-degree-of-freedom robotic crane, known as TETRA [1]) and we needed to create a 3-D visualization of TETRA's operations at a remote site [7]; see Figure 1. Since TETRA's controller

used a different O/S (Microsoft NT) than our visualization platform (an SGI workstation), and the two were in different buildings, a network-based protocol was a clear choice.

Since our immediate need was relatively simple (to update position and orientation of a single item), the resulting "proto-protocol" was quite simple. It was a strict request/response system, built on TCP (streaming) sockets. It used a small set of fixed-format messages (3 messages, all using ASCII data representations.) Communications were strictly host-to-host (as opposed to distributed), but could be opened and closed at run-time. We implemented tetSock in C, on three different platforms: Microsoft Windows NT, UNIX (SUN and SGI systems), and (later) on a real-time O/S, VxWorks.

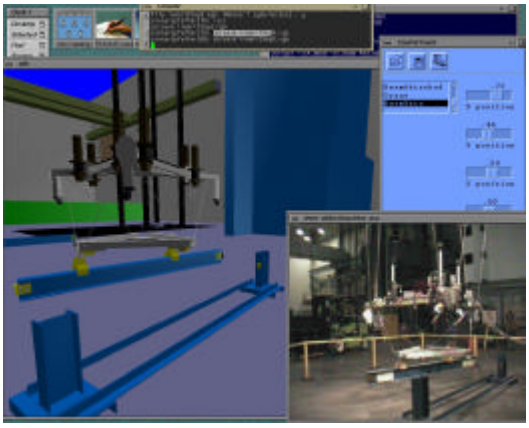

Figure [1] shows the 3-D visualization of a robotic crane during an assembly operation with a steel I-beam. The inset video image shows the correspondence between the model and the real system. The real-time data driving the visualization subsystem came from TETRA's real-time controller, via the tetSock proto-protocol.

Even this minimal protocol has helped with subsystem reuse. Using tetSock, we were able to adapt our visualization system from TETRA (crane updates from an NT system) to a vehicle tracking system (all-terrain-vehicle updates from a VxWorks system) very easily. With a more-carefully-designed protocol, leverage from re-use should become very powerful.

3.1 Lessons learned from tetSock

- Even a very limited networked protocol (like tetSock) helps with system integration.
- *Ad-hoc* protocols are limiting; use/adapt existing protocols if possible.
- Careful, explicit definitions are essential (e.g., which way does X point, is it fixed in world coordinates?)
- Pre-defined messages are somewhat limiting, but (designed correctly) are a workable choice.
- We can achieve significant functionality with a relatively small number of messages: i.e., who, what, where, when.
- Individual host-to-host connections is a poor topology; a central server would be better; a true distributed system, better yet.
- Cross-platform solutions (e.g. not tied to specific O/S or language) are important.
- Temporal issues matter.
- Quality of data will matter.
- Security will matter.

4.0 OVERVIEW OF IEEE 1278

IEEE Standard 1278.1a-1998 is a well specified, post ballot IEEE specification of network data formats and standard practices for distributed interactive simulations (DIS). Originally developed in cooperation with the Department of Defense to support battlefield simulation, the standard has gained support in the general simulation community. The standards covers form and content of messages (termed network protocol data units, PDUs) and the interactions between applications running in the simulation. Fixed data format PDUs are broadcast to participating applications. Management is dynamic and fully distributed (no master controlling application).

In IEEE 1278, entities provide the basic unit of operation. An example of an entity would be a tank. Tanks move, shoot, emit radio waves, are struck by ordinance, and other actions that may require interaction with other entities. Entities represent physical objects in a simulation; at any one time, only one participating application has control of an entity. Applications can control more than one entity. Simulation management functions allow control of entities to be passed between applications.

The PDUs covered in IEEE 1278 are divided into several domains; of specific interest to construction-automation are the domains dealing with entity information/interaction, simulation and management, synthetic environments, and live entity information/interaction. To support interoperability, the format of each type of PDU is fixed, and general notices are sent via multicast UDP packets. Specific PDUs provide the means to communicate answers to the basic questions of who (unique ID), what (event type), where (coordinates in a common frame), and when (time stamp keyed to a common, coordinated clock.) Specific

attention was paid to the development of physical state of an entity. Entity state in IEEE 1278 includes position (translation from the origin), orientation (rotation about axes), articulation (of appendages, where appropriate), and rates of change of these quantities.

IEEE 1278 provides a means of communicating terrain (termed synthetic environment in IEEE 1278) between participating computers, and the ability to modify the synthetic environment on the fly during a simulation. For construction applications, this maps well to earth moving.

The live entity support in IEEE 1278 provides a mechanism where, for example, a real tank could participate in a hybrid live/simulated exercise. The primary purpose of the live entity protocols is to conserve bandwidth to and from these live entities.

Additionally, IEEE 1278 provides a full suite of management functions to provide simulation level management. Examples of such management include mechanisms for running more than one simultaneous exercise and determination for when a delinquent entity is to be removed from a simulation. Simulation management is distributed to all participating applications, with no central server or boss applications. The standard is rich in functions to manage the exchange of information about and between entities. For example, when entities collide, the standard provides a well-developed mechanism for dealing with that collision. Extensions to the set of PDUs are handled robustly by the IEEE 1278 to enable extensions of functional behavior. We plan to take advantage of this feature in LiveView.

4.1 Related Implementations

The DIS-Java-VRML Working Group of the VRML Web 3D consortium [8] is working on the conjunction of IEEE 1278 (DIS), with Java language and the VRML 3-D graphics file format. The focus of this working group is to complete a freely available reference Java implementation of the DIS protocol (in contrast to the current proprietary implementations.) Another goal is to produce a set of recommended practices for mapping between DIS and VRML worlds. The focus of this group is still simulation oriented.

4.2 Beyond Simulation

An essential activity in construction automation is the collection of measurements from field agents (machines, people, or man-machine teams at construction sites.) Although IEEE 1278 provides a mechanism for communicating some information about field agents (live entity protocol), it does not provide a way to communicate the data resulting from the use of sensors that observe the state of the world, as opposed to the state of the entity itself. Also, within IEEE 1278 there is not currently a way to make use of that observed data to update the model of the world in the simulation (e.g. sending an updated terrain model to all participating applications). The purpose of LiveView is to establish a set of standard practices and message protocols to incorporate observed data into a DIS. This is a proposed extension to IEEE 1278.

Both IEEE 1278 and the DIS-Java-VRML working group are focusing on simulation. To these applications, an active sensor is only modeled in terms of whether it can be detected by other entities already simulated in the exercise (and then attacked). However, in construction metrology, a field entity may discover a new entity. Results of sensors that observe the external world, active or passive, are not supported by IEEE 1278. What is needed is a way to report these observations, and potentially offload expensive sensory data analysis from a field observer to a more powerful system.

In line with the rest of IEEE 1278, the LiveView extensions need to be independent of language or operating system. In the future, it is envisioned that vendors would develop products that communicated using this specification. To enable the greatest forward flexibility, this specification should not limit options in realizing implementations.

5.0 OVERVIEW OF LIVEVIEW

LiveView was inspired by the process by which the oil industry provides oil and gasoline to its consumers. A data factory (oil producing company) collects data. Then transmits this mostly unprocessed data (oil tankers transporting crude) to a data interpreter (oil refinery). The interpreter analyses the data to produce IEEE 1278 PDUs (gasoline) to be transmitted to the visualization system (your car.)

5.1 A Simple Example

In a simple case, the data factory is capable of all the necessary sensor processing to create PDUs. Consider a field agent that provides a state update (who/what/where/when) for a static object encountered, for example, reading the bar code from a girder on a construction site; see Figure 2.

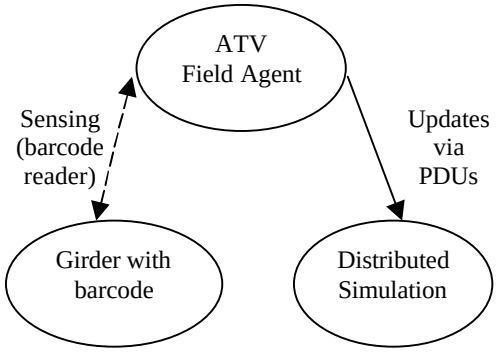

Figure [2]. In this simple case, the field agent can interpret the data from the sensor observing the object to generate PDUs to describe the girder to the simulation.

After the field agent has identified the object and determined the object's position, it is necessary to broadcast any updates to the data consumers. In this case, the facilities for entity creation and managing entity information, provided in IEEE 1278, provide a common data exchange to enable communication between the field agent and the data consumers.

5.2 Complex Case

In some cases, observed data may need significant processing to yield information about entities in the simulation. For example, an entity may need to be recognized by its shape – a significant computation. Another example is the generation of a terrain from a large number of laser range measurements over a field of view (a LIDAR raster scan of the ground); see Figure 3.

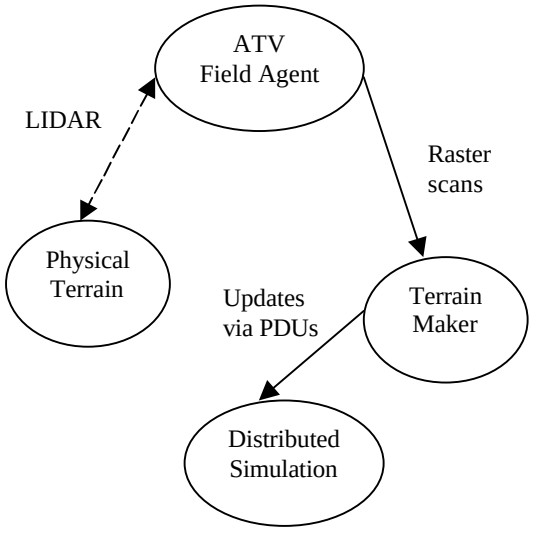

Figure [3]. In this more complex case, an external application is needed to interpret the raster scans and generate the resulting PDUs.

A data factory in this model may be a fix-mounted scanning LIDAR station or a mobile one. The purpose of these factories is to produce raw data in some form, potentially in a proprietary format. In some cases, some local processing or fusion of sensory data may be possible, such as combining GPS data with the range to a target object to provide the global position of the object. In the general case however, the data set may be too large, or the algorithms too expensive for a local system to perform. In such a case, the data set must be packaged and sent to an application that will perform a more detailed analysis of the data set and create the PDUs necessary to update the model of the world.

The data-interpreter application serves as a black box that receives data from one or more factories and produces IEEE 1278 compliant PDUs. Following our example, the location of multiple scanners combined with their respective scans of a construction site provides the input necessary to generate a terrain map of the construction site. In this case, the data interpreter(s) provide scan registration and perform the actual meshing of the data into terrain PDUs.

This proposed structure provides a mechanism for accommodating vendor specific solutions. If a vendor uses a customized reader/sensor, then the vendor could provide an interpreter to provide a mapping between the vendor's proprietary system and this modified DIS standard.

6. FUTURE PLANS

The creation of a LiveView reference implementation of the extensions is an immediate goal of this work. The target platforms for this reference are Microsoft Windows and POSIX (UNIX). Windows is being developed as the reference platform for site display. POSIX platforms will be providing database and network support. Additionally, the mobile sensor platforms run VxWorks, a real-time, POSIX conformant operating system.

Development of standard practice documents for applying LiveView to construction automation and metrology tasks is a related task. These formal documents will provide system integrators with a complete picture of how a working system is put together. Additionally, such a document provides vendors of specific systems a model to follow for how their products could be used.

Another issue we seek to address is to be able to provide "Quality of Data" estimates with measured phenomena. Currently, due to the nature of simulation, if an entity reports a position, there is no reason to question the potential error in that

message. However, field methods for measuring position have limited accuracy. The data quality might be represented with a simple confidence-band, or a more complex function. Issues in how to present and transport information of this nature needs to be investigated further.

Although LiveView covers reporting the results of sensory activities between simulated and non-simulated entities, remote control of live entities is not presently being addressed.

Finally, the issue of security and access rights in LiveView will have to be addressed. Current research in security for distributed systems needs to be evaluated and applied to the LiveView system. This work will become critical before control of live entities is transacted over the system

ACKNOWLEDGEMENTS & NOTICES

The authors wish to thank Fred Proctor for his initial implementation of tetSock.

Certain trade names and company names are mentioned in this paper; in no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products are necessarily the best available for the purposes discussed. All trademarks, servicemarks, etc. are property of their respective holders.

REFERENCES

[1] Bostelman, R., Albus, J. Dagalakis, N. Jacoff, A. "RoboCrane: An Advanced Concept for Large Scale Manufacturing," proceedings of the Association for Unmanned Vehicle Systems International Conference, July 1996, Orlando, FL.

[2] Hoque, R., CORBA 3, IDG Books Worldwide, Inc., Foster City, CA, 1998.

[3] IEEE, “IEEE Std. 1278-1a-1998, Standard for Distributed Interactive Simulations – Application Protocols,” IEEE Customer Service, Piscataway, NJ 0855, (800) 678-IEEE, http://stdbbs.ieee.org/ (refers to 1995 ver.)

[4] Lee, K., Schneeman, R., “A Standardized Approach for Transducer Interfacing: Implementing IEEE-P1451 Smart Transducer Interface Draft Standards”, *Proceedings of SENSORS Conference ’96*, Philadelphia, PA, Helmers Publishing, October 22-24, 1996, pp. 87-100.

[5] Pardo-Castellote, G.. *Experimental Integration of Planning and Control for a Intelligent Manufacturing Workcell*. Ph.D. thesis, Stanford University, Department of Electrical Engineering, Stanford, CA 94305, June 1995, Chapter 4. (available from university microfilms or via download from http://sun-valley.stanford.edu/bib/arlpub.html)

[6] Pfeffer, Lawrence E., "Wireless Networking for Integration of Real-Time Construction Metrology Systems," in the proceedings of the 14th International Symposium on Automation and Robotics in Construction (ISARC 14), Pittsburgh, PA, June, 1997.

[7] Stone, W., Reed, K,. Chang, P., Pfeffer, L., Jacoff, A., "NIST Research toward Construction Site Integration and Automation," ASCE Journal of Aerospace Engineering, Vol. 12, No. 2, April 1999, pp. 50--57.

[8] Web3D consortium http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/dis-java-vrml.html, mirrored at http://www.stl.nps.navy.mil/dis-java-vrml/